SOFE 3980U
Software Quality

Assignment 2

Name: Daniyal Khan (100750029)

Due Date: March 24th ,2022

Problem A Github Link: (https://github.com/100750029/Software-Quality-Assignment1)

Static Analysis Techniques:

An example of a static analysis technique is program slicing. Program slicing is focusing on a specific selection of variables within a program. A program slice refers to the sections of the program that are relevant to the subset of variables. There are four types of program slicing:

- Forward:
  - A forward slice across a program for a given statement X contains all statements that are affected by X.

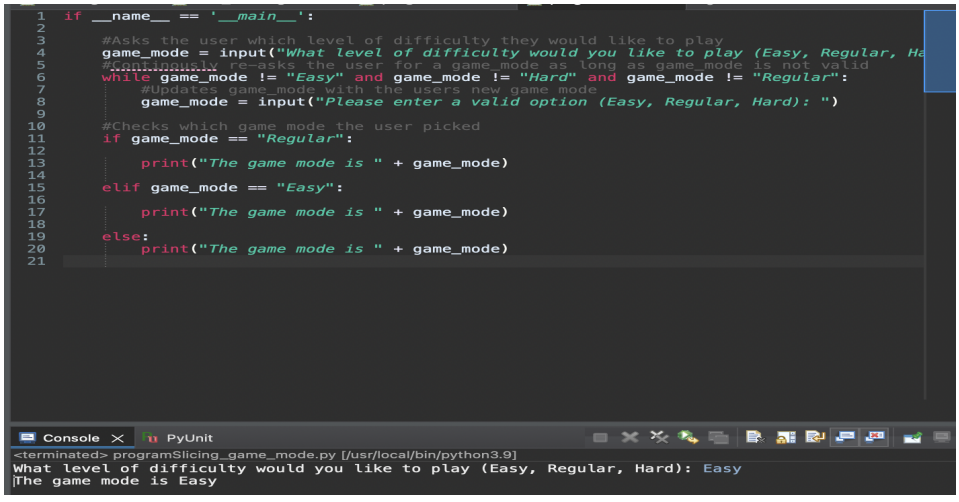
- Backward:
  - A backward slice of a program for a given statement X contains all statements that affect whether control reaches X as well as all statements that affect the value of variables that occur in X.

- Static:
  - A static program slice is calculated symbolically, that is, without taking specific data values into account.

- Dynamic:
  - A dynamic program slice is created using specific data values.

For my program, I chose to do a mix of the following techniques listed above.

Program Slicing:



For the first section, I applied program slicing to a variable called **game_mode**. The variable was only used in the main function and is used to select the difficulty of the game. I replaced the variables, SIZE and TRIES, with a print statement as they set the length of the word and number of attempts you get, and as such aren't necessary. The print statement outputs the difficulty you've selected.

```
def create_code(characters, length):

    '''
    Returns a list the length of length (int) with any
    random single characters from characters (str).

    >>> create_code("Danival", 4)
    ['D', 'i', 'y', 'l']
    >>> create_code("Hello", 3)
    ['H', 'H', 'H']
    '''

    #Creates the list that will store the code
    code = []
    #Adds a random letter from characters to code length amount of times
    for i in range(length):
        code.append(random.choice(characters))

    return code

if __name__ == '__main__':

    #Asks the user which level of difficulty they would like to play
    game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
    #Continously re-asks the user for a game_mode as long as game_mode is not valid
    while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
        #Updates game_mode with the users new game mode
        game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")

    #Checks which game mode the user picked
    if game_mode == "Regular":
        # the number of pegs in the answer for the game mode "Regular"
        SIZE = 4
        # the number of guesses the user gets for the game mode "Regular"
        TRIES = 10
    elif game_mode == "Easy":
        # the number of pegs in the answer for the game mode "Easy"
        SIZE = 3
        # the number of guesses the user gets for the game mode "Easy"
        TRIES = 12
    else:
        # the number of pegs in the answer for the game mode "Hard"
        SIZE = 5
        # the number of guesses the user gets for the game mode "Hard"
        TRIES = 8

    # the letters allowed representing the colours
    # green, red, blue, yellow, orange, purple
    VALID_CHARS = 'WORDLE'

    #Creates the secret code using the function create_code()
    secret_code = create_code(VALID_CHARS, SIZE)

    #Runs as many times as the users number of tries
    for i in range(TRIES):
        #Asks the user for their guess
        guess = input("Please enter guess number " + str(i + 1) + " of length " + str(SIZE) + " using the letters " + VALID_CHARS + ": ")
        #Converts the users guess into a list so that it can be compared to secret_code
        guess = list(guess)
        #Checks if the users guess matches the code (if they have won)
        if guess == secret_code:
            print ("Congratualtions! It took you " + str(i + 1) + " guesses to find the code.")
            #Breaks the for loop so that the user is not asked to guess again (the game has ended)
            break

    #The secret_code will be given if the user looses because they have no tries left
    #(the for loop finished and the user was unable to guess the right code)
    if guess != secret_code:
        print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")
```

.

```
Console  PyUnit
<terminated> programSlicing_secret_code.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Hard
Please enter guess number 1 of length 5 using the letters WORDLE: WOR
Please enter guess number 2 of length 5 using the letters WORDLE: WWW
Please enter guess number 3 of length 5 using the letters WORDLE: WORDE
Please enter guess number 4 of length 5 using the letters WORDLE: RRRRR
Please enter guess number 5 of length 5 using the letters WORDLE: WRONG
Please enter guess number 6 of length 5 using the letters WORDLE: THE
Please enter guess number 7 of length 5 using the letters WORDLE: SAMPLE
Please enter guess number 8 of length 5 using the letters WORDLE: W
I'm sorry you lose. The correct code was REOWO.
```

Next I chose to do program slicing for the variable **secrete_code**. This variable involves the create_code function so we need to include it in our program. When I run the program it prompts me for the difficulty and then asks me to enter a guess. It doesn't give me any hints or clues because those variables aren't reliant on the secrete_code variable. It also doesn't check is the user guess is valid and it outputs the correct answer after you've reached the maximum number of tries.

```
1  if __name__ == '__main__':
2
3      # the letters allowed representing the colours
4      VALID_CHARS = 'WORDLE'
5
6      #Creates the secret code using the function create_code()
7      secret_code = VALID_CHARS
8
9      #Asks the user for their guess
10     guess = input("Please enter guess using the letters " + VALID_CHARS + ": ")
11
12     #Checks if the users guess matches the code (if they have won)
13     if guess == secret_code:
14         print ("Congratulations! you guessed the code.")
15
16     else:
17         print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")
18
```

```
Console  PyUnit
<terminated> programSlicing_VALID_CHARS.py [/usr/local/bin/python3.9]
Please enter guess using the letters WORDLE: WORDLE
Congratulations! you guessed the code.
Relaunch with the same configura
with focus on editor)
```

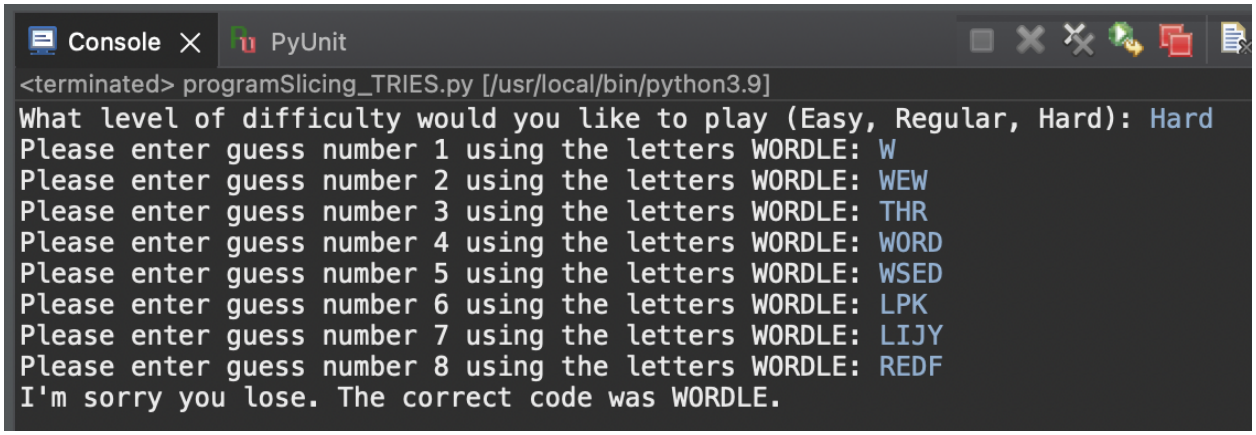
For the variable **VALID_CHARS,** I chose to do dynamic program slicing. I made it so the secret_code was already set to "WORDLE" which meant we didn't need any clues or hints. It doesn't check if the guess is valid either. Also, the program didn't need to create a code for the user to guess. The user just needs to input the guess and then the program will output if you got it correct or not.

```python
def valid(user_guess, valid_characters, guess_size):

    '''
    Returns True if all the characters in user_guess (list) are in
    valid_characters (str) and the length of user_guess is equal
    to guess_size (int).

    >>> valid(['H', 'E', 'L', 'L', 'O'], "HELOYT", 5)
    True
    >>> valid(['S'], "S", 3)
    False
    >>> valid(['G', 'R', 'R', 'Y'], "GRBYOP", 4)
    True
    '''

    #Checks if the length of user_guess is equal to guess_size
    if len(user_guess) == guess_size:
        #Goes through each character in user_guess
        for character in user_guess:
            #Returns False if the character is not found in valid_characters
            if character not in valid_characters:
                return False
    else:
        return False

    return True

if __name__ == '__main__':

    #Asks the user which level of difficulty they would like to play
    game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
    #Continously re-asks the user for a game mode as long as game mode is not valid
    while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
        #Updates game_mode with the users new game mode
        game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")

    #Checks which game mode the user picked
    if game_mode == "Regular":
        # the number of pegs in the answer for the game mode "Regular"
        SIZE = 4
        # the number of guesses the user gets for the game mode "Regular"
        TRIES = 10
    elif game_mode == "Easy":
        # the number of pegs in the answer for the game mode "Easy"
        SIZE = 3
        # the number of guesses the user gets for the game mode "Easy"
        TRIES = 12
    else:
        # the number of pegs in the answer for the game mode "Hard"
        SIZE = 5
        # the number of guesses the user gets for the game mode "Hard"
        TRIES = 8

    # the letters allowed representing the colours
    # green, red, blue, yellow, orange, purple
    VALID_CHARS = 'WORDLE'

    #Runs as many times as the users number of tries
    for i in range(TRIES):
        #Asks the user for their guess
        guess = input("Please enter guess number " + str(i + 1) + " of length " + str(SIZE) + " using the letters " + VALID_CHARS + ": ")
        #Converts the users guess into a list so that it can be compared to secret_code
        guess = list(guess)
        #Continously asks the user for a valid guess as long as their guess is not valid
        while not valid(guess, VALID_CHARS, SIZE):
            #Updates guess to store the users new guess
            guess = input("Please re-enter a valid guess of length " + str(SIZE) + " using the letters " + VALID_CHARS + ": ")
            #Converts the users guess into a list so that it can be compared to secret_code
            guess = list(guess)
```

```
Console ×  PyUnit
programSlicing_valid.py [/usr/local/bin/python3.9]
What level of difficulty would you like to play (Easy, Regular, Hard): Hard
Please enter guess number 1 of length 5 using the letters WORDLE: THE
Please re-enter a valid guess of length 5 using the letters WORDLE: WOR
Please re-enter a valid guess of length 5 using the letters WORDLE: WORD
Please re-enter a valid guess of length 5 using the letters WORDLE: WORDL
Please enter guess number 2 of length 5 using the letters WORDLE: WORDLE
Please re-enter a valid guess of length 5 using the letters WORDLE:
```

Next I did static program slicing for the function **valid**. This function includes the objects user_guess, valid_characters, and guess_size. This program checks if the user's guess is valid based on the SIZE and VALID_CHARS. The function asks the users for a difficulty then the user must input a guess. The function checks if the guess is valid based on the constraints.

```python
if __name__ == '__main__':

    #Asks the user which level of difficulty they would like to play
    game_mode = input("What level of difficulty would you like to play (Easy, Regular, Hard): ")
    #Continously re-asks the user for a game_mode as long as game_mode is not valid
    while game_mode != "Easy" and game_mode != "Hard" and game_mode != "Regular":
        #Updates game_mode with the users new game mode
        game_mode = input("Please enter a valid option (Easy, Regular, Hard): ")

    #Checks which game mode the user picked
    if game_mode == "Regular":
        # the number of guesses the user gets for the game mode "Regular"
        TRIES = 10
    elif game_mode == "Easy":
        # the number of guesses the user gets for the game mode "Easy"
        TRIES = 12
    else:
        # the number of guesses the user gets for the game mode "Hard"
        TRIES = 8

    # the letters allowed representing the colours
    # green, red, blue, yellow, orange, purple
    VALID_CHARS = 'WORDLE'

    #Creates the secret code using the function create_code()
    secret_code = VALID_CHARS

    #Runs as many times as the users number of tries
    for i in range(TRIES):
        #Asks the user for their guess
        guess = input("Please enter guess number " + str(i + 1)  + " using the letters " + VALID_CHARS + ": ")
        #Converts the users guess into a list so that it can be compared to secret_code
        guess = list(guess)
        if guess == secret_code:
            print ("Congratualtions! It took you " + str(i + 1) + " guesses to find the code.")
            #Breaks the for loop so that the user is not asked to guess again (the game has ended)
            break

    #The secret_code will be given if the user looses because they have no tries left
    #(the for loop finished and the user was unable to guess the right code)
    if guess != secret_code:
        print("I'm sorry you lose. The correct code was " + ("").join(secret_code) + ".")

```



Finally I did static program slicing for a variable called **TRIES**. This variable assigns the user a number of attempts based on the difficulty they've selected. I manually created a secrete_code because we didn't need the create_code function. Also we didn't need to include the SIZE of the word because we are only checking if the program gives you the correct number of tries. The program doesn't check if the guess you've entered is valid and outputs the correct answer after you've reached the maximum number of tries.

Problem B

Dynamic Analysis:

Instrumentation adds measurement probes to the code so that it may be observed while it runs. It is possible to accomplish this on numerous levels. There are various ways for various levels. Each approach has distinct overheads and levels of accuracy. To evaluate the run time of functions, I created a function called elapsed_time:

```
import time
import datetime


def elapsed_time(func):

    def inside_function(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        remove_fully_correct_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return remove_fully_correct_function

    return inside_function

```

This function uses the import time and datetime to get the elapsed time of a function. It gets the elapsed time by subtracting the end time by the start time. It then displays the time by month and date.

Instrumentation:

```python
import time
import datetime
import random

def elapsed_time(func):

    def inside_fuection(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        create_code_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return create_code_function

    return inside_fuection

@elapsed_time
def create_code(characters, length):

    '''
    Returns a list the length of length (int) with any
    random single characters from characters (str).

    >>> create_code("Danival", 4)
    ['D', 'i', 'y', 'l']
    >>> create_code("Hello", 3)
    ['H', 'H', 'H']
    '''

    #Creates the list that will store the code
    code = []
    #Adds a random letter from characters to code length amount of times
    for i in range(length):
        code.append(random.choice(characters))

    return code

create_code_function = create_code(['D', 'A', 'N', 'I', 'Y', 'A', 'L'], 4)
print("The create_code function will output: " + str(create_code_function))

```

```
Console × PyUnit
<terminated> dynamicAnalysis_create_code.py [/usr/local/bin/python3.9]
create_code start time: 2022-03-24 02:43:50.506638
create_code end time: 2022-03-24 02:43:50.506865
create_code Elapsed time: 00:00:00.000227
The create_code function will output: ['N', 'D', 'D', 'L']
```

The create_code function returns a list the length of length (int) with any random single characters from characters (str). This code is created using the characters “DANIYAL” with the length of 4. The code that is created is “NDDL” and the function runs with an elapsed time of 227 micro seconds.

```python
import time
import datetime

def elapsed_time(func):

    def inside_function(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        remove_fully_correct_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return remove_fully_correct_function

    return inside_function

@elapsed_time
def remove_fully_correct(list1, list2):

    '''
    Returns a list that has all the characters in list1
    except for those that are in the same position as the
    same character in list2.

    >>> remove_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
    ['R', 'O', 'P']
    >>> remove_fully_correct(['A','B','O','D'], ['D','B','A','D'])
    ['A', 'O']
    >>> remove_fully_correct(['B', 'B', 'B'], ['A', 'B', 'B'])
    ['B']
    '''
    #Creates the new list
    new_list = []
    #Goes through each index value in list1
    for i in range(len(list1)):
        #Checks if the letter at the index i in list1 is
        #not the same letter at the index i in list 2
        if list1[i] != list2[i]:
            #Adds that letter to new_list
            new_list.append(list1[i])

    return new_list

remove_fully_correct_function = remove_fully_correct(['h','e','l','l','o'], ['h','l','l','h','h'])
print("The remove_fully_correct function will output: " + str(remove_fully_correct_function))
```

```
Console  PyUnit
<terminated> dynamicAnalysis_remove_fully_correct.py [/usr/local/bin/python3.9]
remove_fully_correct start time: 2022-03-24 03:23:02.537911
remove_fully_correct end time: 2022-03-24 03:23:02.538139
remove_fully_correct Elapsed time: 00:00:00.000228
The remove_fully_correct function will output: ['e', 'l', 'o']
```

The remove_fully_correct function returns a list that has all the characters in list1 except for those that are in the same position as the same character in list2. The two lists that the function is comparing are list1= “hello” and list2=“hllhh”. The function returns with a list “elo” and runs with an elapsed time of 228 micro seconds.

```python
import time
import datetime

def elapsed_time(func):

    def inside_function(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        find_fully_correct_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return find_fully_correct_function

    return inside_function

@elapsed_time
def find_fully_correct(answer, guess):

    '''
    Returns a list containing the letter "b" for each
    colour in guess (list of strs) that is in the same position as the
    same colour in answer (list of strs).

    >>> find_fully_correct(['G', 'B', 'R', 'Y'], ['G', 'R', 'R', 'O'])
    ['b', 'b']
    >>> find_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
    ['b']
    >>> find_fully_correct(['P', 'G', 'R', 'O', 'Y'], ['Y', 'G', 'O', 'O', 'Y'])
    ['b', 'b', 'b']
    '''
    #Creates a list that will store the "b" clues
    correct_b = []
    #Goes through each index value in answer
    for i in range(len(answer)):
        #Checks if the colour at the index i in guess is
        #the same colour at the index i in answer
        if guess[i] == answer[i]:
            #Adds a "b" to correct_b
            correct_b.append("b")

    return correct_b

find_fully_correct_function = find_fully_correct(['h','e','l','l','o'], ['h','h','h','h','h'])
print("The find_fully_correct function will output: " + str(find_fully_correct_function))
```

```
Console  PyUnit
<terminated> dynamicAnalysis_find_fully_correct.py [/usr/local/bin/python3.9]
find_fully_correct start time: 2022-03-24 03:05:03.134292
find_fully_correct end time: 2022-03-24 03:05:03.134518
find_fully_correct Elapsed time: 00:00:00.000226
The find_fully_correct function will output: ['b']
```

The find_fully_correct function returns a list containing the letter "b" for each color in the guess (list of strs) that is in the same position as the same color in the answer (list of strs). The two lists the function is comparing are answer = “hello” and guess = “hhhhh”. The function returns a list “b” and runs with an elapsed time of 226 micro seconds.

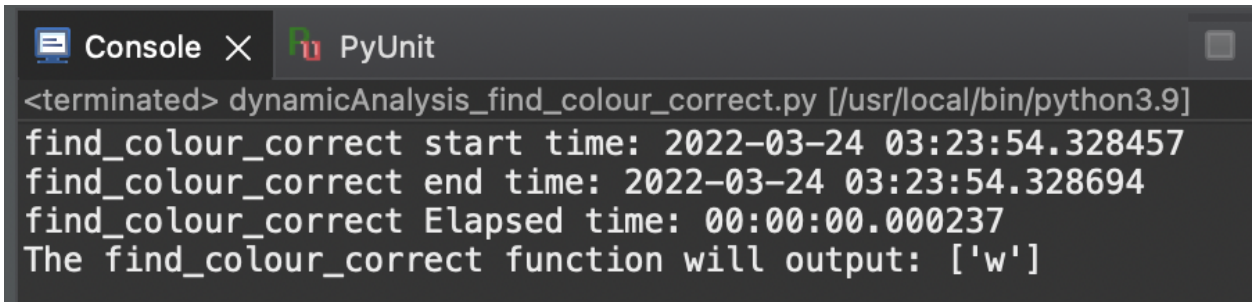
```python
import time
import datetime

def elapsed_time(func):

    def inside_fuection(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        valid_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return valid_function

    return inside_fuection

@elapsed_time
def valid(user_guess, valid_characters, guess_size):

    '''
    Returns True if all the characters in user_guess (list) are in
    valid_characters (str) and the length of user_guess is equal
    to guess_size (int).

    >>> valid(['H', 'E', 'L', 'L', 'O'], "HELOYT", 5)
    True
    >>> valid(['S'], "S", 3)
    False
    >>> valid(['G', 'R', 'R', 'Y'], "GRBYOP", 4)
    True
    '''
    #Checks if the length of user_guess is equal to guess_size
    if len(user_guess) == guess_size:
        #Goes through each character in user_guess
        for character in user_guess:
            #Returns False if the character is not found in valid_characters
            if character not in valid_characters:
                return False
    else:
        return False

    return True

valid_function = valid(['h','h','l','l','h'], ['h','e','l','l','o'], 5)
print("The valid function will output: " + str(valid_function))
```

```
Console × PyUnit
<terminated> dynamicAnalysis_valid.py [/usr/local/bin/python3.9]
valid start time: 2022-03-24 03:13:00.528763
valid end time: 2022-03-24 03:13:00.528976
valid Elapsed time: 00:00:00.000213
The valid function will output: True
```

The function valid returns True if all the characters in user_guess (list) are in valid_characters (str) and the length of user_guess is equal to guess_size (int). The function inputs are user_guess = ‘hhllh’, valid_characters = ‘hello’, and guess_size = 5. The function returns True and runs with an elapsed time of 213 micro seconds.

```python
import time
import datetime

def elapsed_time(func):

    def inside_fuection(*args):

        start_time = time.time()
        print(func.__name__, 'start time:', datetime.datetime.utcfromtimestamp(start_time))
        find_colour_correct_function = func(*args)
        end_time = time.time()
        print(func.__name__, 'end time:', datetime.datetime.utcfromtimestamp(end_time))
        time_elapsed = datetime.datetime.utcfromtimestamp(end_time-start_time)
        print(func.__name__, 'Elapsed time:', time_elapsed.strftime('%H:%M:%S.%f'))
        return find_colour_correct_function

    return inside_fuection

def remove_fully_correct(list1, list2):

    '''
    Returns a list that has all the characters in list1
    except for those that are in the same position as the
    same character in list2.

    >>> remove_fully_correct(['R', 'R', 'O', 'P'], ['P', 'R', 'R', 'O'])
    ['R', 'O', 'P']
    >>> remove_fully_correct(['A','B','O','D'], ['D','B','A','D'])
    ['A', 'O']
    >>> remove_fully_correct(['B', 'B', 'B'], ['A', 'B', 'B'])
    ['B']
    '''
    #Creates the new list
    new_list = []
    #Goes through each index value in list1
    for i in range(len(list1)):
        #Checks if the letter at the index i in list1 is
        #not the same letter at the index i in list 2
        if list1[i] != list2[i]:
            #Adds that letter to new_list
            new_list.append(list1[i])

    return new_list

@elapsed_time
def find_colour_correct(answer, guess):

    '''
    Returns a list containing the letter "w" for each
    character in guess (list of strs) that is also in answer (list of strs)
    but isn't in the right position.

    >>> find_colour_correct(['Y','P','G','G'],  ['G','P','O','R'])
    ['w']
    >>> find_colour_correct(['O', 'P', 'P', 'R'], ['O', 'R', 'P', 'P'])
    ['w', 'w']
    >>> find_colour_correct(['Y','P','G'],  ['G','G','O'])
    ['w']
    '''
    #Creates the list that will store all the "w" clues
    correct_w = []
    #Removes all the colours that are in the right position from both answer and guess
    new_guess = remove_fully_correct(guess, answer)
    new_answer = remove_fully_correct(answer, guess)
    #Goes through each colour in new_guess
    for colour in new_guess:
        #Checks if the colour is in new_answer
        if colour in new_answer:
            #Adds a "w" to correct_w
```

```
69              #Adds a "w" to correct_w
70              correct_w.append("w")
71              #Removes that colour from new_answer so that it isn't checked twice
72              new_answer.remove(colour)
73
74      return correct_w
75
76  find_colour_correct_function = find_colour_correct(['y','p','g'], ['g','g','o'])
77  print("The find_colour_correct function will output: " + str(find_colour_correct_function))
78
```



The function find_colour_correct returns a list containing the letter "w" for each character in guess (list of strs) that is also in answer (list of strs) but isn't in the right position. The function inputs are answer = ‘ypg’ and guess = ‘ggo’. The function returns ‘w’ and runs with an elapsed time of 237 micro seconds.

Challenges Faced:

Now I will go over some of the challenges I faced throughout this assignment:

- The first challenge I faced was learning how to slice a program. I learned about program slicing through the lecture slides and online tutorials. I figured out the types of slicing you can do and started to implement them.

- The next challenge I faced was deciding which level I should apply instrumentation. I decided I would apply instrumentation on the functions. I came to the conclusion that it would be interesting seeing how long each function takes to run.

- Another challenge I was faced with was learning how to apply instrumentation. I solved this problem by watching videos on youtube. I watched a video and learned I can get the elapsed time of a function by creating a function which subtracts the end time with the start time.